

CS351 Fall 2008, Final Project

Image Genome Project

MondoSoft, Inc.

STATUS Specification and requirements document

VERSION 1.2

DATE Nov 18, 2008

0 Changelog

Version 1.0 Initial release. Nov 18, 2008.

Version 1.1 Fixed due dates. Nov 20, 2008.

Version 1.2 Added examples for image and color generation. Nov 20, 2008

1 Summary

Thanks to the overwhelming success of Webcrawler, Mondosoft is delving into the seedy world of image generation. Since content creation is expensive and labor intensive, your employers believe the future of image generation is automatic. Having over-heard the buzzword "Genetic Algorithm" at a charity brunch, Mondosoft's CTO has tasked you and your team with developing a system capable of evolving images toward those most pleasing to the unwashed masses. This system should be able to generate images that humans like and copy already successful images.

Contents

0	Changelog	1
1	Summary	1
2	Group/Individual Effort	3
3	Java Language Level	3
4	Definitions	3
5	Requirements	4
5.1	The Parser	4
5.2	Expression Tree	6
5.3	Evaluator	6
5.4	User Interface	7
5.4.1	User Mode	8
5.4.2	Target Mode	8
5.4.3	Next Generation	8
5.5	Gene Pool	9
6	Quantitative Requirements	9
7	Deliverables	10
7.1	Milestone 1: JUnit Test Suites	11
7.2	Milestone 2: Parser and Expression Tree Classes	12
7.3	Milestone 3: Genome Evolution Suite	13
7.4	Final Rollout: Full Image Genome Software Suite and Gene Pool Server	14
8	Due Dates	15

2 Group/Individual Effort

This project is an *group* programming project. Refer to the CS351 Fall 2008 Syllabus and FAQ for details on what this means. Programming teams should consist of no more than 4 people and no fewer than 3.

3 Java Language Level

You **MUST** write this project in Java 1.5.x (a.k.a., “the Java 5 platform”) or a later version. Your code **MUST** support generic classes where required by the required project interfaces.

4 Definitions

The following definitions will be used in this document:

Genome A mathematical expression, that when evaluated on a grid of (x,y) locations creates an image.

Phenome An image generated using a mathematical equation.

Infix expression Notation in which the operator separates its operands. E.g. $(a + b) * c$. Infix notation requires the use of brackets to specify the order of evaluation

Fitness Function A function that determines the survival of a Genome, this function may be based on a score (numerical quality) or simply user preference.

MAY A requirement that the product can choose to implement if desired. Can also indicate a choice among acceptable alternatives (e.g., “The program **MAY** do x, y, or z.” indicates that the choice of behavior x, y, or z is up to the designer.)

MUST A requirement that the product must implement for full credit.

MUST NOT A behavior or assumption that must not be violated. Violating a **MUST NOT** restriction will result in a penalty on the assignment.

SHOULD A requirement that is recommended, but not required. The designer may violate a **SHOULD** requirement, but should be prepared to explain why.

SUBMISSION ARCHIVE The archive file that the designer turns in for each milestone. The **SUBMISSION ARCHIVE** **MAY** be a tarball (`.tar.gz`), jar file, or ZIP archive. It **MUST NOT** be a proprietary or non-portable format. The **SUBMISSION ARCHIVE** file **MUST** be named `lastname_p2[milestone](version).[extension]` where `[milestone]` indicates which milestone is being submitted (M1, M2, M3, or R) and `[extension]` indicates the type of archive being submitted (e.g., `.tar.gz`, `.tgz`, `.jar`, etc.). The `(version)` string is optional; if present, it indicates which version of a milestone is being submitted, in case the designer wishes to submit a revised version of the milestone.

Examples of valid SUBMISSION ARCHIVE names are `lane_p2m1.tar.gz`, `lane_p2m3_v2.jar`, and `lane_p2R_v0.zip`.

The contents of the SUBMISSION ARCHIVE MUST be a top-level directory with the same name as the SUBMISSION ARCHIVE file, omitting the extension. Thus, for the above examples, the SUBMISSION ARCHIVE would contain directories named `lane_p2m1`, `lane_p2m3_v2`, and `lane_p2R_v0`, respectively. If the SUBMISSION ARCHIVE is a jar file, it MAY contain additional jar administrative files (such as the manifest). Beyond that, the SUBMISSION ARCHIVE MUST NOT contain any other top-level files. All submitted files MUST be located within the top-level directory.

UNRECOVERABLE ERROR An error condition from which recovery is impossible. The program MUST produce an error message describing the condition and then cleanly halt. The program MUST NOT crash, core-dump, display a stack trace, dump a raw exception to the screen, or corrupt any stored data.

5 Requirements

This section describes the elements that MUST be developed as part of this project. The designer MAY also choose to implement additional Java source files, programs, and/or shell scripts in support of the following items. This section only describes the general performance requirements for each element; for specific deliverable requirements, please refer to Section 7.

The Image Genome Project suite comprises four program components: `Parser`, `Evaluator`, `User Interface`, and `Gene Pool`.

`Parser` is responsible for reading an `Infix` expression and generating an `Expression Tree` data structure for use in other program components. The `Evaluator` takes a parse tree and generates a `Phenome` image of some specified dimensions and quality. The `User Interface` is responsible for displaying `Genome` equations and `Phenome` images, handling user interactions, driving image evolution, and interfacing with the `Gene Pool`. The `Gene Pool` is a remote server that stores and provides access to successful `Genes` for future use.

In addition to the “performance” code of this project, you will also develop a complete set of test cases and test code for all elements of the project. Some of this MUST be in `JUnit`, but it will probably prove necessary to have external (e.g., shell script) test suites as well.

The designer MAY choose any package naming convention (including the default package) for this project. If a root package other than the default is chosen, it SHOULD be

```
edu.unm.cs.[yourname].cs351.p2
```

If the choice of package affects how the programs are invoked, the `README.TXT` document MUST specify this.

5.1 The Parser

The parser is a software component (class) that take a text string that contains a valid infix expression and generates an `Expression Tree` data structure. The parser MUST accept the following tokens:

- () Open and close parenthesis, used to indicate order of operations
- 0-9.0-9** Numeric floating point precision constants in ASCII format
- x,y** Variables x and y indicate Cartesian coordinates
- +** Binary addition
- Binary subtraction or unitary negation
- *** Binary multiplication
- /** Binary division
- pow(a,b)** Binary exponent, a to the b power
- log(a)** Unitary logarithm, log base e of a
- exp(a)** Unitary exponentiation, same as $\text{pow}(e,a)$
- sin(a)** Unitary sine, sine of a
- asin(a)** Unitary arc-sine, arc-sine of a
- cos(a)** Unitary cosine, cosine of a
- acos(a)** Unitary arc-cosine, arc-cosine of a
- tan(a)** Unitary tangent, tangent of a
- atan(a)** Unitary arc-tangent, arc-tangent of a
- atan2(a,b)** Binary arc-tangent taking two terms, arc-tangent of a,b

Your implementation MAY implement any other operations you choose so long as they are well documented and DO NOT conflict with the set above.

An example valid infix expression is

```
(x+y)*pow(x, sin(y))/log(atan2(x+y, exp(10.234)))
```

When the parser encounters an invalid expression (*e.g.* unmatched parens or unallowed tokens) it MUST throw an `InvalidParseExpression` exception that MUST be captured and handled by the software component calling the parser. The Class implementing the `Parser` MUST be called `ExpParser`, and MUST include a function called `parseInfix` which takes a string containing an infix expression and returns an `Expression Tree` if the string is successfully parsed. This Class MAY implement additional operations at the designers option.

5.2 Expression Tree

The `Parser` component needs to return a data structure that represents the expression so that it can be used to generate images. This data structure, called the `Expression Tree`, **MUST** be able to evaluate the expression for any given value of the variables `x` and `y`. This data structure **SHOULD** be implemented as a tree. All nodes of the tree **MUST** implement the following interface:

```
Interface ExpressionNode<T> {
    public T evaluate(T x, T y);
};
```

Where `T` is either `float` or `double`. The function `evaluate` takes the `x` and `y` coordinates (image locations) and returns the value of the expression. The `Expression Tree` **SHOULD** be specialized for different kinds of expression tokens, *i.e.* constants, variables, and operators. Note that constants and variables will always be leaf nodes in the tree and operators such as `+`, `*`, `sin()`, `cos()` will always be internal nodes.

The `Expression Tree`'s `evaluate` function **MUST** be thread safe, *i.e.* capable of being called by multiple threads simultaneously.

The `Expression Tree` will be used extensively in the `User Interface` software components and will need to be efficiently manipulated (changed) in the image manipulation process. Your design **SHOULD** take these manipulations into account. Evaluation of the `Expression Tree` should be linear in the number of nodes it contains.

5.3 Evaluator

The evaluator is a `Class` that supports the transformation of an expression genome into an image phenome. The class must support a function that takes an expression tree and returns an image. The evaluator `Class` **MUST** support the following interface

```
Interface Evaluator<T> {
    public BufferedImage
        generateImage(ExpressionNode<T> expression,
                    int xdim, int ydim,
                    T xstart, T ystart,
                    T xend, T yend );
};
```

The function `generateImage()` takes an expression tree, the dimensions of the image to generate (`xdim`, `ydim`), and the starting and ending (`x,y`) coordinate for evaluation (`xstart`, `ystart`) and (`xend`, `yend`). The function returns a `BufferedImage` object, which is found in `java.awt.image.BufferedImage`. You **MAY** implement additional functions that simplify the function's signature.

Example pseudo code for evaluating an image is shown below,

```
float x = xstart;
float xinc = (xend-xstart)/xdim;
float y = ystart;
float yinc = (yend-ystart)/ydim;
for(int i=0; i<ydim; ++i)
    for(int j=0; j<xdim; ++j)
    {
        float tmp = expression.evaluate( x + j*xinc, y+ i*yinc );
        img[i][j] = getColor( tmp );
    }
```

where `getColor()` is a function that returns an RGB color for any real value and `img` is a 2D array of RGB colors.

This function **MUST** evaluate the expression tree for each pixel and convert that value into an RGB color. You **MAY** develop any conversion function you like, so long as it generates an RGB color. You **MAY** want to develop multiple methods for color generation. Your system **MUST** be able to evaluate the following color function, shown in pseudo code:

```
float[3]  grayScaleColor( float value )
{
    float[3] color;
    float val = value;
    if( value < 0 ) val = -value;
    for(int i=0; i<3; ++i)
        color[i] = (val/10.0)/(val/10.0+1) * 255;
    return color;
}
```

This function returns a "gray scale" RGB color in the range $[0 - -255]$. Your system **MUST** also implement at least one additional color function, and allow the user to select which color function to use in the User Interface.

The `generateImage` function **MUST** always run on a different thread than the User Interface. You **MAY** adapt this function to generate an image using multiple threads.

5.4 User Interface

The User Interface drives the image evolution in two ways. The first mode, User Mode, allows the user to drive the Fitness Function by selecting images that "survive" the current generation. The second mode, Target Mode, allows the user to specify an image that the evolution will target. The Fitness Function for the second case is the difference between the target image and the current Phenome images. The user interface will also acquire expression genes from the Gene Pool server and submit successful expression genes to the Gene Pool server.

The User Interface **MUST** provide the user with an array of thumbnail images (no less than 16 in a 4 by 4 grid) that represent the current generation. These thumbnail images **MUST NOT** be smaller than 32 by 32 pixels. It **MUST** also provide the user with a larger image (no

smaller than 256 by 256 pixels) when a particular thumbnail is selected. The expression (in infix notation) that generated the image must also be presented with the larger image. When a user selects an image in `User Mode`, the image **MUST** be highlighted and used for genetic crossover/mutation in the next generation. The `User Interface` **MUST** provide a text input widget that allows the user to add an infix expression gene to the current generation. The `User Interface` **MUST** provide a button with the text "Next Generation" that generates a new array of images using genetic crossover and mutation (applicable in `User Mode`).

The `User Interface` **MUST** have the following menu items

Save State Allows the current generation of equation `Genomes` to be saved to the local disk.

Save Image Saves the currently selected image as a .png (Ping image file format)

Load Image Loads a .png image (and any other formats you choose to implement) for use in `Target Mode`

Evolve Automatically generates images toward the target image (when a target image has been loaded)

Help Generates a help window that describes the application. It **MUST** identify the authors of the program, version number, and provide a brief description of how the application works.

You **MAY** choose to implement additional features in the `User Interface` at your discretion. These features must be documented and included in the help screen.

5.4.1 User Mode

In `User Mode`, each generation is created from previous generation using the set of user selected images. The user can select any number of images to "survive" for genetic crossover and mutation. When the user does not select any images, the next generation **CAN** be generated using random expression genes.

5.4.2 Target Mode

In `Target Mode` a new generation is generated by first comparing each image in the current generation to the target image. Fifty percent of the images in the current generation will be evolved using genetic crossover and mutation to create the next generation. The `Fitness Function` of a image **SHOULD** be a score generated by computing the difference between a `Phenome` image and the target image, where lower scores indicate a higher likelihood of survival.

5.4.3 Next Generation

The next generation of images is generated by creating new expression genomes from the surviving set. This is accomplished in two ways: Genetic Crossover, and Mutation. Half of the new genomes should be generated using genetic crossover and half should be generated using Mutation.

Genetic Crossover is achieved by randomly selecting a sub-tree from one surviving genome and replacing it with a randomly selected subtree from another surviving genome.

Mutation is achieved by randomly selecting a node in a genome and replacing it with a randomly selected new node. Mutation must also have the ability to delete a node with some probability (start with 10%).

5.5 Gene Pool

The `Gene Pool` is a server application that stores and delivers genome expressions. This server **MUST** accept requests for genome expressions and accept submissions of genome expressions. All genome expressions **MUST** be accompanied with the integer generation number that genome was created in. The `Gene Pool` server **MUST** select expressions for delivery with a probability proportional to the generation number of the available expressions. For instance if the highest generation number any equation in the `Gene Pool` is 9, than an expression with a generation number of 2 would be selected with a 20% probability. Specifically, the probability of selection is $g/(max_g + 1)$, where g is the generation number and max_g is the largest generation number in the gene-pool. The `Gene Pool` **MUST** accept infix expressions accompanied by its generation number.

A request for genome equations from the client **MUST** be strings of the form:

```
Request Expressions: <number of equations requested>
```

When this string is recieved by the `Gene Pool` server, it **MUST** respond by sending the number of requested equations. If the `Gene Pool` does not contain the requested number of equations it **MAY** respond with fewer equations or randomly generated equations. It **MUST NOT** deliver more equations than requested. Equations must be in infix form and separated by newline characters.

A submission of equations from the client **MUST** be strings of the form:

```
Expression Submission <generation number>: <infix expression>
```

The `Gene Pool` server **MUST** save all expressions along with their generation number to file. It **SHOULD** do this automatically. It **MUST** load this file whenever it is started. When no file is found it **MUST** start a new one.

6 Quantitative Requirements

This section describes the performance and behavior requirements for the webalyzer software suite.

1. All programs **MUST NOT** crash, core dump, dump a stack trace, or throw an exception on any input.
2. In the case of a **RECOVERABLE ERROR**, a program **MUST** issue a warning statement and continue processing. The program **MAY** choose to issue the warning statement to standard error, to a log file, or to a user interface element. If the warning is issued to a log file, the log file name and location **MUST** be a user-specifiable parameter to the program (either by command-line command or via a configuration menu).
3. In the case of an **UNRECOVERABLE ERROR**, a program **MUST** issue an error statement and terminate with a non-zero error condition. The program **MAY** use different exit codes to

indicate different error conditions, but such codes **MUST** be documented in the user manual. The error message **MUST** be logged to the same destination that warning messages (from RECOVERABLE ERRORS) are.

4. In the case of any **ERROR**, a program **MUST NOT** delete, corrupt, or damage existing files or any other “stateful” files employed by the program suite.
5. The programs **MAY** provide additional output for debugging purposes, *but* such output must be *disabled by default*. Any program **MAY** provide a command-line switch or a user-interface configuration utility to enable debugging support when desired.
6. The software components **MAY** assume that all valid user input is standard ASCII text in the range. If a program encounters a character outside this range, it **MAY** treat it as a RECOVERABLE or UNRECOVERABLE ERROR or silently ignore it. If such characters are treated as RECOVERABLE or ignored, they **MUST NOT** disrupt the otherwise normal functioning of the program.
7. All user documentation **MUST** be grammatically correct and include correct spelling and usage. (You *will* be graded, in part, on the quality of your writing.)
8. The designer **MUST** document any areas in which her or his software suite does not meet this specification. **WARNING!** The grade penalty will be higher if the instructors discover an undocumented program shortcoming or bug than if it is documented up front.

7 Deliverables

This section describes the content to be delivered at each stage of the project (three milestones and a final rollout). Each deliverable is a superset of the previous one – it **MUST** include all of the materials from the previous deliverable as well as the new materials. Milestone 2 and Rollout **MAY** contain updates to the previous deliverables. E.g., if webgraph was not fully functional at Milestone 2, a revised version **MAY** be submitted at Milestone 3 or Rollout. Doing so will not change the grade on the previous delivery, but it will contribute to a better grade on the current deliverable. Similarly, more test cases can be submitted in Milestone 2 or later, etc.

If updates to a past deliverable are included, the `README.TXT` file **MUST** indicate which components (including code, documentation, tests, etc.) have been modified. For the deadlines of these stages, please refer to Section 8.

For each submission, the designer **MAY** assume that the submitted code will be executed in an environment with the following items freely available (not requiring further reference or configuration):

- The complete, standard, Javasoft Java JDK, version 1.5.x, including the `java`, `jar`, and `javadoc` executables, and the complete JDK 1.5.x library suite.
- The `JUnit` library, version 1.4.x.
- Standard shell-scripting interpreters, including `sh`, `bash`, `perl`, and `python`.

The designer MAY assume that all of the above are already available on the appropriate `PATH` and `CLASSPATH` variables.

Any other programs, support libraries, or configurations necessary to run/test the submitted code MUST be approved in advance and documented in the submission. All submissions must be capable of being compiled and run standalone. In particular, the submissions MUST NOT assume that the code is being loaded and run in Eclipse, NetBeans, or any other IDE. The submission also MUST NOT assume that any instructor code *other* than the Support Code `.jar` file is available at runtime.

7.1 Milestone 1: JUnit Test Suites

The first project component due is a set of JUnit tests for the `Parser`, `Expression Tree`, `User Interface`, and `Gene Pool` classes. These tests MUST cover all methods in both classes and include both reasonable, unreasonable, and erroneous inputs. Edge cases and corner cases MUST also be examined. Every test MUST be fully documented and describe what is being tested, how, and expected behavior. For Milestone 1, it is *not* necessary to provide tests of asymptotic time and space performance, although those will be assessed in a later milestone.

Note that implementations of these classes are NOT required for this milestone, but you will need to consider the abstract design of the classes and test any additional methods required by your design.

The deliverables for this milestone are:

UML Design UML Class Diagrams for each of the software components.

UML Sequence UML Sequence Diagrams for the `User Interface` and its interactions with the other software components

TestParser.java The JUnit test suite for the `Parser` class.

TestExpressionTree.java The JUnit test suite for the `Expression Tree` class(es).

TestUI.java The JUnit test suite for the `User Interface` class(es)

TestGenePool.java The JUnit test suite for the `Gene Pool` server

Other Java source files Any other supporting code files necessary to compile, load, and use the JUnit test files.

API documentation The handin MUST also include the full, compiled Javadoc documentation for all Java source files. This documentation MUST include full descriptions of every public or protected method, field, sub-class, enclosed class, or constructor employed by any of these Java files. This documentation hierarchy MUST be included in a sub-directory named `api-doc/` within the submission tarball package.

At the designer's option, this submission MAY also include:

BUGS.TXT This file documents any known outstanding bugs, missing features, performance problems, or failures to meet specifications of your submission. Note that the penalty for such problems will be smaller if they're fully documented here than if the instructors discover them independently. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

README.TXT This file includes any other notes or documentation necessary to use or understand the submission that are not already included in other documentation. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

All materials for Milestone 1 **MUST** be packaged in a SUBMISSION ARCHIVE and mailed to *both* Prof. Kniss and TA Godinez by the due date.

7.2 Milestone 2: Parser and Expression Tree Classes

The second component of the project is the complete implementation of the `Parser` and `Expression Tree` classes. These **MUST** be complete and functional Java classes that implement the required interfaces. These classes **MUST** implement all methods specified in their respective interfaces and **MUST** support all non-method functionality either explicitly or implicitly described in this specification document (e.g., serializability of the expression tree).

These classes **MAY**, in addition, implement additional methods or functionalities not specified by the interfaces. Any such functionality **MUST**, however, be fully documented in Javadoc API documentation.

You **MUST** also provide a "mock-up" of the `User Interface`. This is a non-functioning demonstration of the user interface. It **SHOULD** be implemented using swing, with all required menu items and window components.

The deliverables for this milestone are:

Parser The Java code file for the `Parser` implementation.

Expression Tree The Java code file for the `Expression Tree` implementation.

User Interface mock-up A non-functioning user interface prototype

Other Java source files Any other supporting code files necessary to compile, load, and use the webgraph and crawlstate files.

JUnit source files Test code files for all of the above. This set of files **MUST** contain at least the tests submitted in Milestone 1, but it **MAY** contain additional tests. The original tests **MAY** be revised to improve them or make them more complete.

API documentation The handin **MUST** also include the full, compiled Javadoc documentation for all Java source files. This documentation **MUST** include full descriptions of every public or protected method, field, sub-class, enclosed class, or constructor employed by any of these Java files. This documentation hierarchy **MUST** be included in a sub-directory named `api-doc/` within the SUBMISSION ARCHIVE.

Empirical demonstration code Any Java files necessary to execute the empirical demonstration of runtime. This MAY also include any shell scripts or other support code necessary to complete the runtime tests. These materials MUST be located in a directory named `analysis/src` within the top-level directory of the SUBMISSION ARCHIVE.

At the designer's option, this submission MAY also include:

BUGS.TXT This file documents any known outstanding bugs, missing features, performance problems, or failures to meet specifications of your submission. Note that the penalty for such problems will be smaller if they're fully documented here than if the instructors discover them independently. If present, this file MUST be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

README.TXT This file includes any other notes or documentation necessary to use or understand the submission that are not already included in other documentation. If present, this file MUST be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

All materials for Milestone 2 MUST be packaged in a SUBMISSION ARCHIVE and mailed to *both* Prof. Kniss and TA Godinez by the due date.

7.3 Milestone 3: Genome Evolution Suite

The third component of the project is the full Image Genome program and supporting code and documentation. In addition to the complete executable code for the `User Interface`, this submission also includes user documentation sufficient for a naive user (one who has never looked at the code and has no intention of doing so) to execute and use program and understand its output. This submission also includes an example or tutorial of the system's usage, using screen shots etc.

If the designer has made changes to code already submitted in Milestone 2 (e.g., `webgraph` or `crawlstate`), those changes MUST be documented in the `README.TXT` file.

The deliverables for this milestone are:

The User Interface code All Java files necessary to compile and execute the `User Interface` program. This will require fully working `Parser` and `Expression Tree` software components

Other Java source files Any other supporting code files necessary to compile, load, or use the `Crawler` program.

JUnit source files Test code files for all of the above. This set of files MUST contain at least the tests submitted in Milestone 2, but it MAY contain additional tests. The original tests MAY be revised to improve them or make them more complete.

API documentation The handin MUST also include the full, compiled Javadoc documentation for all Java source files. This documentation MUST include full descriptions of every public or protected method, field, sub-class, enclosed class, or constructor employed by any of these Java files. This documentation hierarchy MUST be included in a sub-directory named `api-doc/` within the SUBMISSION ARCHIVE.

User documentation A document describing how to use the image evolution program. This document **MUST** also include at least one example of how to run the program and how to interpret output. This document **MUST** be named `USERDOC.[extension]` and **MUST** be located in a directory named `doc/`, located off the top-level directory within the SUBMISSION ARCHIVE. It **MAY** be a plain text, HTML, PDF, or PostScript document (with the appropriate `extension`). It **MUST NOT** be a Microsoft Word or other nonportable format document.

Example output Screen shots of the program in use, with interesting image examples. This file **MUST** be named `EXAMPLE-OUTPUT.pdf` and **MUST** be located in the `doc/` sub-directory.

At the designer's option, this submission **MAY** also include:

BUGS.TXT This file documents any known outstanding bugs, missing features, performance problems, or failures to meet specifications of your submission. Note that the penalty for such problems will be smaller if they're fully documented here than if the instructors discover them independently. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

README.TXT This file includes any other notes or documentation necessary to use or understand the submission that are not already included in other documentation. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

All materials for Milestone 3 **MUST** be packaged in a SUBMISSION ARCHIVE and mailed to *both* Prof. Kniss and TA Godinez.

7.4 Final Rollout: Full Image Genome Software Suite and Gene Pool Server

The final component of the project is the complete implementation of the Image Genome suite, including all code previously developed in earlier milestones. The rollout also includes the Gene Pool Server as well as full testing and user documentation.

If the designer has made changes to code already submitted in Milestones 1–3, those changes **MUST** be documented in the `README.TXT` file.

The deliverables for this milestone are:

GenePool.java The Java source code file for the Gene Pool Server program.

Other Java source files Any other supporting code files necessary to compile, load, and use the complete Image Genome suite.

JUnit source files Test code files for all of the above. This set of files **MUST** contain the tests submitted in Milestones 1–3, as well as tests for the new code developed for Rollout. It **MAY** also contain additional tests for Milestones 1–3 code.

API documentation The handin **MUST** also include the full, compiled Javadoc documentation for all Java source files. This documentation **MUST** include full descriptions of every public or protected method, field, sub-class, enclosed class, or constructor employed by any of these Java files. This documentation hierarchy **MUST** be included in a sub-directory named `api-doc/` within the SUBMISSION ARCHIVE.

Target Mode evolution results A report that **MUST** be provided in a portable document format such as PDF, HTML, PostScript, or ASCII. It **MUST NOT** be a Microsoft Word or other non-portable format. This document **MUST** demonstrate a target image and an evolved image that your system attempted to match. This **MUST** include a description of the quality of the match and a justification for the quality of the results.

Image Genome suite user documentation Documentation on how to compile and run all programs in the software suite, including any information about necessary external library support and CLASSPATH information. This **MUST** also include documentation on all command-line options of all programs, expected outputs of all programs, and potential error conditions that the user could encounter while running the program. Finally, this documentation **MUST** include examples of running each program, expected output of that run, and how to interpret the output.

This document **MUST** be named `USERDOC.[extension]` and be located in the directory `doc/` within the top-level directory of the SUBMISSION ARCHIVE. This document **MAY** be a plain text, HTML, PDF, or PostScript document (with the appropriate `extension`). It **MUST NOT** be a Microsoft Word or other nonportable format document.

At the designer's option, this submission **MAY** also include:

BUGS.TXT This file documents any known outstanding bugs, missing features, performance problems, or failures to meet specifications of your submission. Note that the penalty for such problems will be smaller if they're fully documented here than if the instructors discover them independently. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

README.TXT This file includes any other notes or documentation necessary to use or understand the submission that are not already included in other documentation. If present, this file **MUST** be located in a sub-directory named `doc/`, off of the top-level directory within the SUBMISSION ARCHIVE.

All materials for Rollout **MUST** be packaged in a SUBMISSION ARCHIVE and mailed to *both* Prof. Kniss and TA Godinez by the due date.

8 Due Dates

Please refer to the class syllabus for the policy on late handins.

Nov 18, 2:00 PM, MDT Final Project specification v. 1.0 handed out.

Nov 26, 12:01 AM, MDT Milestone 1 due.

Dec 3, 12:01 AM, MDT Milestone 2 due.

Dec 10, 12:01 AM, MDT Milestone 3 due.

Dec 17, 12:01 AM, MDT Final Rollout.